Αναφορά 3ης Εργαστηριακής Άσκησης στο μάθημα
Δομές Δεδομένων & Αρχείων
Καπελώνης Χαρίλαος, ΑΜ: 2020030058

Α) Υλοποίηση ζητούμενων δομών δεδομένων και αρχείων σε Java:

Λίγα λόγια για τον κώδικα:

Το πρόγραμμα υλοποιεί τις ζητούμενες δομές, μία στη μνήμη (B+Tree) και δύο στον δίσκο (B Tree, AVL Tree). Οι αντίστοιχες κλάσεις βρίσκονται στα πακέτα memory και disk. Το πακέτο charilaospackage αφορά κλάσεις βοηθητικές ή/και κοινές για τα δύο άλλα πακέτα. Επίσης έχει την κλάση Main και την κλάση Test. Στις δομές του δίσκου, αυτό που διαφοροποιείται είναι ότι τα δεδομένα (integers) που εισάγουμε αποθηκεύονται σε ένα αρχείο, αλλά με συγκεκριμένο τρόπο. Αρχικά, μετατρέπουμε τον integer σε bytes (4 bytes) και τα προσθέτουμε σε έναν πίνακα με μέγεθος 128 ή 256 bytes. Μια σελίδα δίσκου DataPage (στην οποία θα είναι αποθηκευμένα τα δεδομένα) μπορεί να αποθηκευτεί σε αρχείο. Μπορούμε και να διαβάζουμε και να γράφουμε από/σε ένα αρχείο, DataPages.
Η κλάση FileController βοηθάει στην ανάγνωση των αρχείων που δόθηκαν. Συνολικά δόθηκαν τέσσερα αρχεία – ένα με 10^6 κλειδιά (για την κατασκευή των δέντρων) και τρία από 100 κλειδιά το καθένα (για εισαγωγές, αναζητήσεις και διαγραφές).
Το ζητούμενο της άσκησης ήταν για κάθε δομή να υλοποιηθεί η συνάρτηση insert(int), search(int) και delete(int) έτσι ώστε να μετρήσουμε την απόδοση των δομών. Για κάθε δομή υπήρξαν 10 test cases: για 10^5, $1X10^5$, $2X10^5$,…, $10X10^5$ αρχικά εισαγόμενα στοιχεία έγιναν 100 εισαγωγές, αναζητήσεις και διαγραφές. Έτσι, δημιουργήθηκαν τα αντίστοιχα διαγράμματα.
Σημειώνεται ότι το πρόγραμμα στον υπολογιστή μας απαιτεί 22303 δευτερόλεπτα (περίπου 6 ώρες) για να ολοκληρωθεί.

1) B+Tree στον δίσκο:

Ο κώδικας βρέθηκε στο Github και για την συγκεκριμένη δομή τροποποιήθηκε βάσει των αναγκών της άσκησης. Συγκεκριμένα, ο δημιουργός είχε υλοποιημένες τις συναρτήσεις (πράξεις) εισαγωγής, αναζήτησης, διαγραφής και, επιπλέον, υπήρχαν τα πεδία pageReads και pageWrites τα οποία αύξαναν μέσα από κλήσεις άλλων συναρτήσεων. Για την άσκηση αθροίσαμε τα δύο νούμερα μετά το τέλος κάθε απαιτούμενης πράξης και τα επιστρέφαμε ως diskAccesses, δηλαδή αριθμό προσβάσεων στη μνήμη. Επίσης, ο δημιουργός είχε έτοιμες τις συναρτήσεις runInsertTrial, runSearchTrial, runDeletionTrials επί των οποίων κάναμε τα απαραίτητα test, πάνω στα ανάλογα κλειδιά.
Αυτό που δεν άλλαξε είναι η βάση της δομής: χρησιμοποιείται η κλάση BplusTree (για το ίδιο το δέντρο), η κλάση TrialClass (για τα τεστ που προαναφέρθηκαν) και οι δύο βοηθητικές κλάσεις BplusConfiguration και BplusTreePerformanceCounter. Στην BplusConfiguration δίνουμε ως ορίσματα το μέγεθος της σελίδας. Για 128 bytes, πρέπει το entrySize να είναι 1, ενώ για 256, 20 bytes.

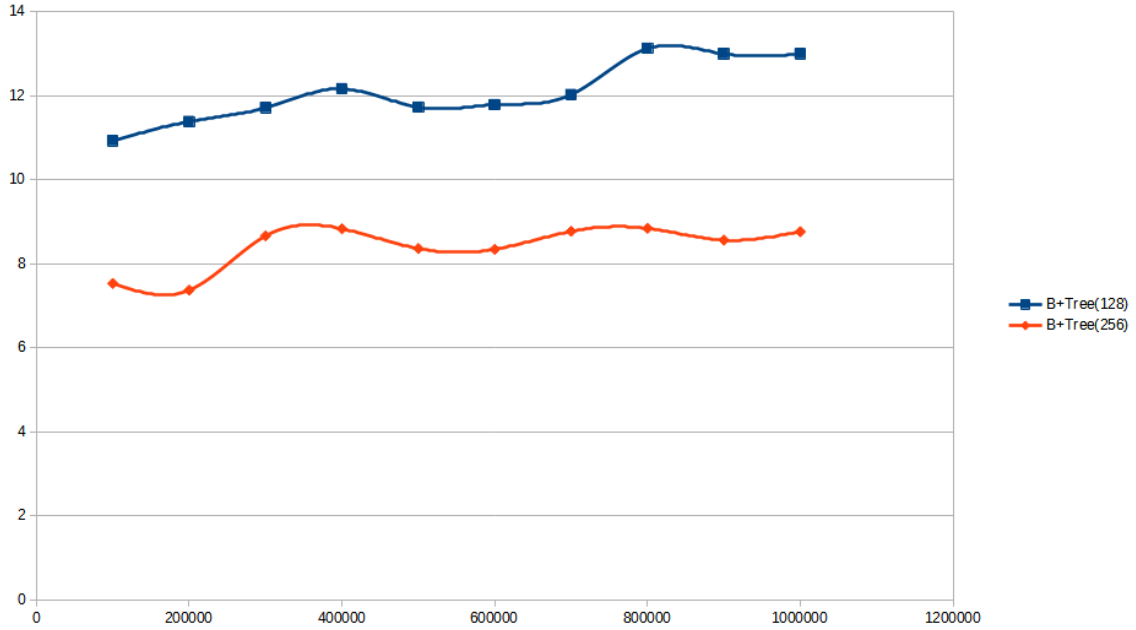
2) B-Tree στη μνήμη:

Ο κώδικας βρέθηκε στο διαδίκτυο και για την συγκεκριμένη δομή τροποποιήθηκε βάσει των αναγκών της άσκησης. Γενικά, η συγκεκριμένη υλοποίηση χρησιμοποιεί την Generic κλάση Btree.java, στην οποία για Key θέτουμε Integer και για Value ένα κενό string, αφού δεν είναι αντικείμενο ενδιαφέροντος της άσκησης. Επίσης μπορούμε να ορίσουμε στον constructor και τον βαθμό του δέντρου. Οι πράξεις κι εδώ είναι έτοιμες: με την put(int, “”) εισάγουμε, με την get(int) αναζητάμε και, τέλος, με την put(int, null) ουσιαστικά διαγράφουμε δεδομένα (βλ. σχόλια δημιουργού “If the value is {@code null}, this effectively deletes the key...”). Εδώ μετρήθηκαν οι συγκρίσεις (if) επικεντρώνοντας το ενδιαφέρον στα for,while-loops με μεγάλο εύρος, τα οποία καθορίζουν τελικά την απόδοση.

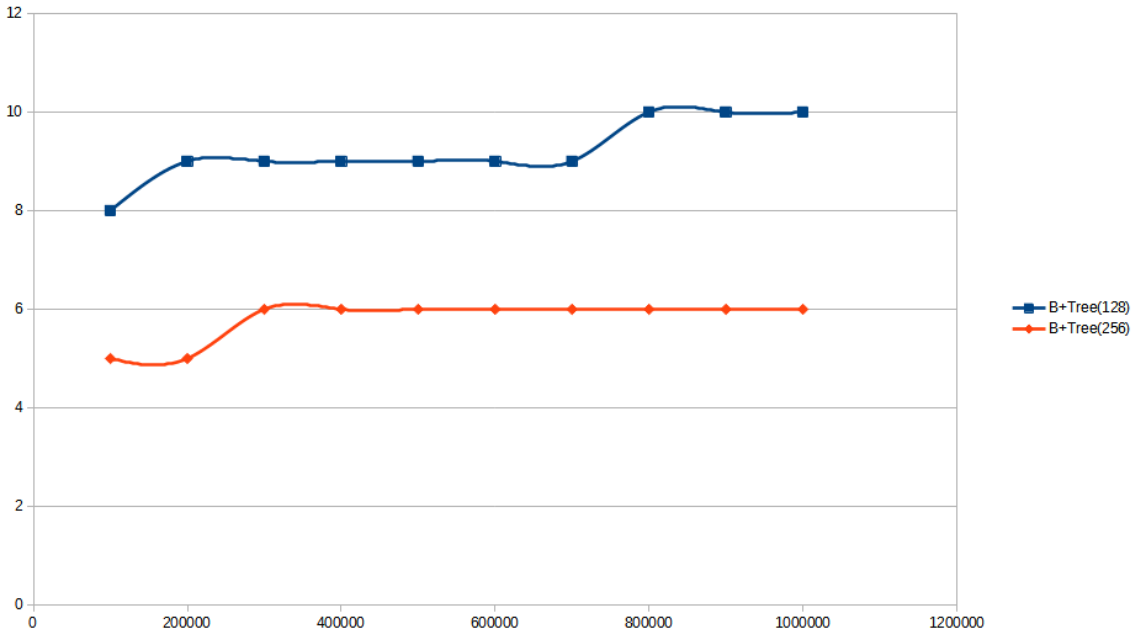
3) AVL-Tree στη μνήμη:

Ο κώδικας βρέθηκε στο διαδίκτυο και για την συγκεκριμένη δομή τροποποιήθηκε βάσει των αναγκών της άσκησης. Ο συγκεκριμένος κώδικας δεν έχει κάποια ιδιαιτερότητα καθώς, αφότου μελετήθηκε, συμπεραίνουμε ότι δεν χρειάζεται κάποια αλλαγή εκτός από την τοποθέτηση αύξησης συγκρίσεων (comparisons++) σε ορισμένα σημεία. Η υλοποίηση αυτή ακολουθεί ένα γνωστό τρόπο γραφής κώδικα (ειδικά δομών δεδομένων): χρησιμοποιεί public wrapper συναρτήσεις [insert(int), delete(int)] οι οποίες καλούν private αναδρομικές. Η find(int), ωστόσο, λειτουργεί με ένα while-loop.

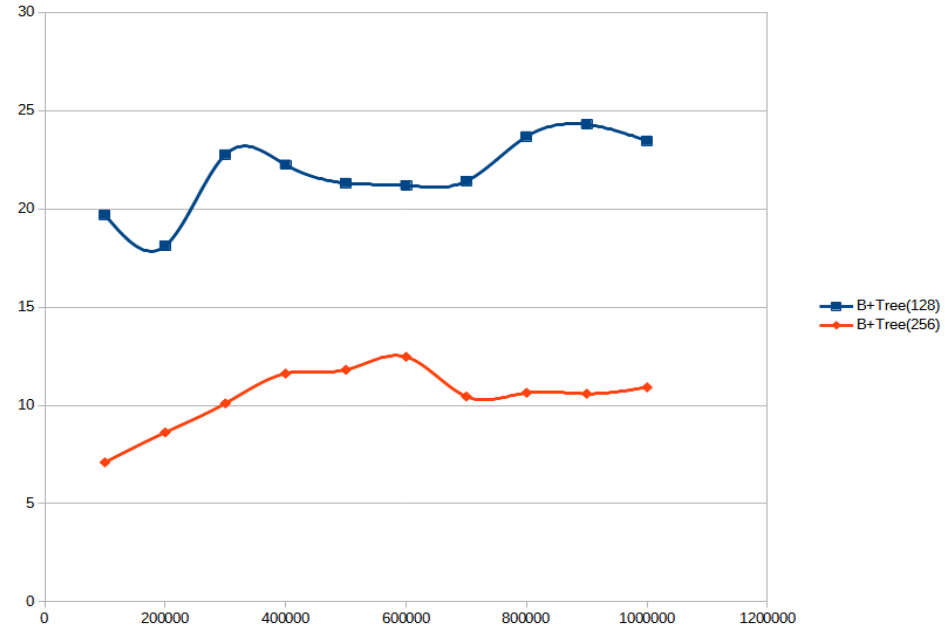
Β) Απόδοση και σύγκριση μεθόδων/ σχολιασμός αποτελεσμάτων:



Διάγραμμα 1: Εισαγωγές B+Tree για μέγεθος σελίδας δίσκου 128 και 256 bytes



Διάγραμμα 2: Αναζητήσεις B+Tree για μέγεθος σελίδας δίσκου 128 και 256 bytes



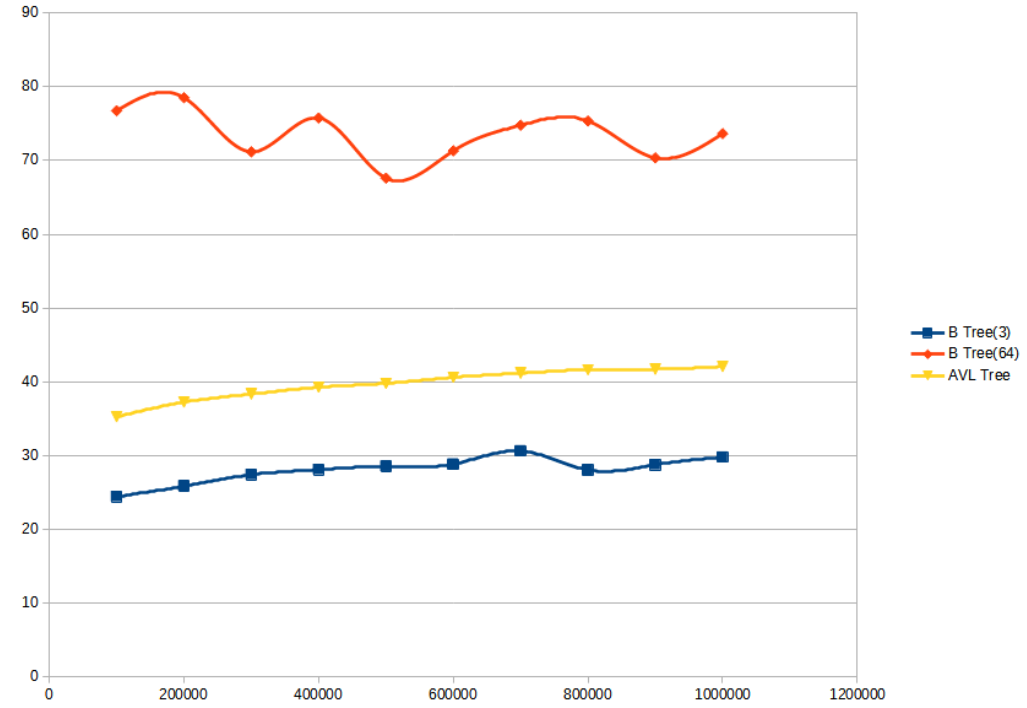
Διάγραμμα 3: Διαγραφές B+Tree για μέγεθος σελίδας δίσκου 128 και 256 bytes

Γενικός σχολιασμός:
Μικρότερο μέγεθος σελίδας δίσκου, σημαίνει ότι λιγότεροι ακέραιοι μπορούν να αποθηκευτούν σε αυτήν, άρα περισσότερες προσβάσεις στον δίσκο για την μεταφορά του ίδιου όγκου δεδομένων από τον δίσκο στη μνήμη. Γι' αυτό βλέπουμε την κατακόρυφη - σχεδόν σταθερή - διαφορά των δύο γραφημάτων.

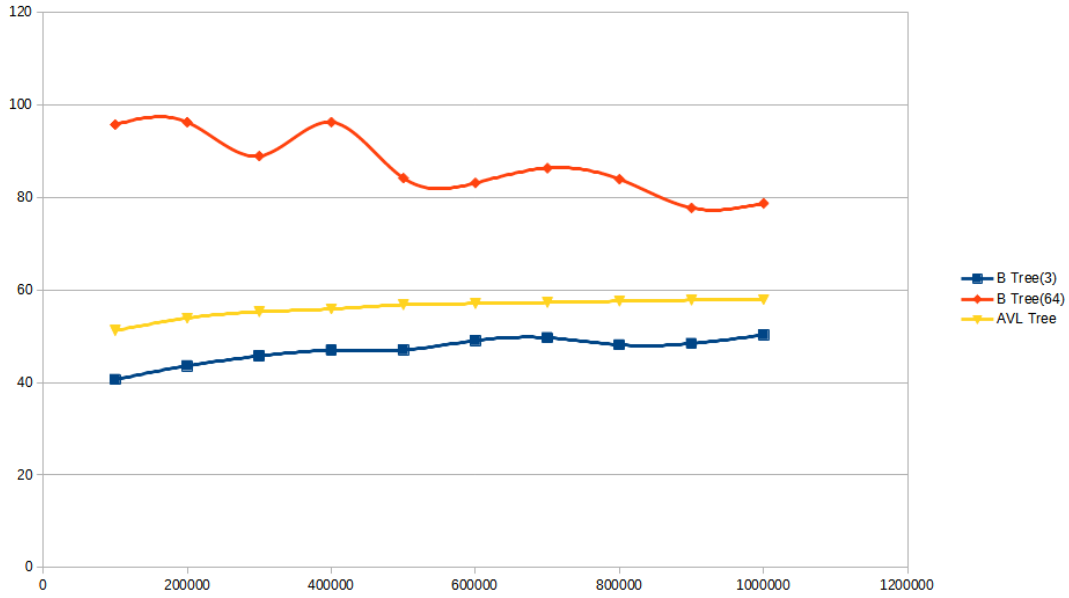
Συγκεκριμένα:
Ο αριθμός συγκρίσεων ανά **εισαγωγή** σε γενικές γραμμές αυξάνεται, ενώ σε κάποια σημεία παρατηρούνται "ανωμαλίες", με την έννοια ότι δεν παίρνουμε μια αύξουσα συνάρτηση. Αυτό οφείλεται στον κώδικα (όχι σε bugs) και στην τυχαιότητα των δειγμάτων μας, μπορεί να τυχαίνει να είναι λιγότερες οι συγκρίσεις λόγω της παρούσας κατάστασης του δέντρου (που είναι self-balancing).

Ο αριθμός των συγκρίσεων ανά **αναζήτηση** αυξάνει, το οποίο είναι λογικό, αφού αναζητάμε κλειδιά ανάμεσα σε περισσότερα στοιχεία. Παρατηρούμε, όμως, ότι αφενός και στα δύο γραφήματα έχουμε πολύ μικρή αύξηση στις συγκρίσεις και αφετέρου ότι οι συγκρίσεις έχουν όμοια μορφή μέχρι τα 700.000 αρχικά εισαγόμενα δείγματα. Φαίνεται αύξηση από 9 σε 10 στην πάνω γραφική (128 bytes μέγεθος σελίδας) ενώ στην κάτω (256 bytes) σταθερά 6. Αυτό είναι λογικό και οφείλεται στην ίδια τη δομή και στα χαρακτηριστικά της. Το ύψος του δέντρου είναι μικρότερο για 256 bytes, άρα έχουμε μικρότερο αριθμό συγκρίσεων.

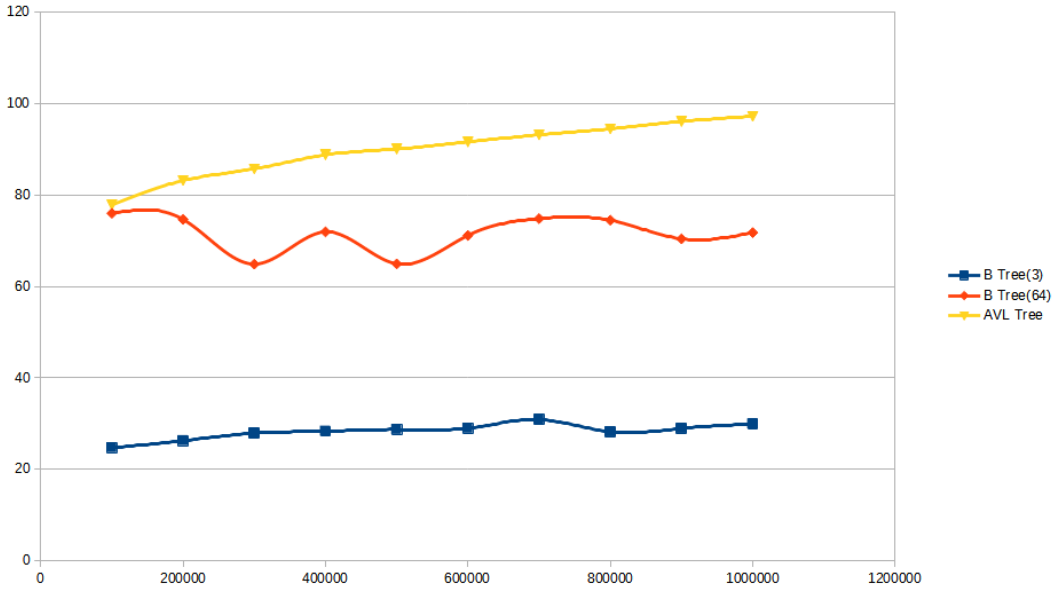
Ο αριθμός των συγκρίσεων ανά **διαγραφή** αυξομειώνεται, αλλά σε κάθε περίπτωση η αρχική με την τελική τιμή έχουν διαφορά. Κι αυτό, όπως και οι εισαγωγές, έχουν να κάνουν με την τυχαιότητα των στοιχείων που επιθυμούμε να διαγράψουμε (και με το αν αυτά υπάρχουν).



Διάγραμμα 4: Εισαγωγές B-Tree (βαθμός 3 και 64) και AVL-Tree



Διάγραμμα 5: Αναζητήσεις B-Tree (βαθμός 3 και 64) και AVL-Tree



Διάγραμμα 6: Διαγραφές B-Tree (βαθμός 3 και 64) και AVL-Tree

Γενικός σχολιασμός:
Παρατηρείται μία αύξηση στον αριθμό συγκρίσεων στο καθώς αυξάνονται τα αρχικώς εισαγόμενα στοιχεία κυρίως στο AVL Tree και στο B-Tree για degree = 3. Όταν ο βαθμός είναι 64 (>>3) ο αριθμός συγκρίσεων αυξομειώνεται, ενώ θα μπορούσαμε να τον χαρακτηρίσουμε απρόβλεπτο.

Συγκεκριμένα:
Ο αριθμός συγκρίσεων ανά **εισαγωγή** σε γενικές γραμμές αυξάνεται, ενώ σε κάποια σημεία παρατηρούνται "ανωμαλίες", με την έννοια ότι δεν παίρνουμε μια αύξουσα συνάρτηση, ιδιαίτερα για το B-Tree(64).

Ο αριθμός των συγκρίσεων ανά **αναζήτηση** αυξάνει, το οποίο είναι λογικό, αφού αναζητάμε κλειδιά ανάμεσα σε περισσότερα στοιχεία. Στο B-Tree(64), έχουμε το μη αναμενόμενο αποτέλεσμα μείωσης τπυ αριθμού των συγκρίσεων. Αυτό έχει να κάνει με το πόσο μεγάλος είναι ο βαθμός του δέντρου.

Ο αριθμός των συγκρίσεων ανά **διαγραφή** αυξομειώνεται. Κι αυτό, όπως και οι εισαγωγές, έχουν να κάνουν με την τυχαιότητα των στοιχείων που επιθυμούμε να διαγράψουμε (και με το αν αυτά υπάρχουν).

Γ) Πηγές:

https://github.com/andylamp/BPlusTree
https://algs4.cs.princeton.edu/code/edu/princeton/cs/algs4/BTree.java
https://www.baeldung.com/java-avl-trees